LABORATORIO 4

NOMBRES DE AUTORES:
IDROBO AVIRAMA SEBASTIÁN (2122637-3743)
SÁNCHEZ LOZANO BRAYAN ANDRÉS (2128974-3743)
CORRALES ARANGO CARLOS DANIEL (2122878-3743)

Presentado a:
Jefferson A. Peña Torres

BASES DE DATOS

UNIVERSIDAD DEL VALLE
INGENIERÍA DE SISTEMAS
CALI
2022

Actividad No. 1

1. $\pi_{A3}(R1) \rightarrow V. \{< q > | \exists t, p(< t, p, q > \in R1)$

Justificación: Esto debido a que en la consulta estamos pidiendo el conjunto de las tuplas "*q"* tales que existen un *"p"* y un *"q"* que pertenezcan a la relación *R1*, siendo *q* las tuplas tomadas equivalentes a *A3.*

2. $\sigma_{A1=15}(R2) \rightarrow II. \{< a, b, c > | < a, b, c > \in R2 \wedge a = 15\}$

Justificación: Esto debido a que en la consulta estamos pidiendo el conjunto de las tuplas *a, b, c* tales que pertenecen a la relación *R2* y sobre las cuales *a* es igual a 15, siendo *a* las tuplas tomadas equivalentes a *A1*.

3. $R1 \cup R2 \rightarrow I. \{< a, b, c > | < a, b, c > \in R1 \vee < a, b, c > \in R2\}$

Justificación: Esto debido a que en la consulta estamos pidiendo el conjunto de las tuplas *a, b, c* tales que pertenecen a *R1*, o que pertenecen a *R2*, siendo la unión de estas dos.

4. $R2 \cap R1 \rightarrow VI. \{< a, b, c > | < a, b, c > \in R1 \wedge < a, b, c > \in R2\}$

Justificación: Esto debido a que en la consulta estamos pidiendo el conjunto de las tuplas *a, b, c* tales que pertenecen a *R1*, y también a *R2*, siendo la intersección de estas dos.

5. $R1 - R2 \rightarrow IV. \{< a, b, c > | < a, b, c > \in R1 \wedge < a, b, c > \notin R2\}$

Justificación: Esto debido a que en la consulta estamos pidiendo el conjunto de las tuplas *a, b, c* tales que pertenecen a *R1*, y que no pertenecen a *R2*, formando así la diferencia entre estas dos.

6.
$\Pi_{A1, A2}(R1) \bowtie \Pi_{A2,A3}(R2) \rightarrow III. \{< a, b, c > | \exists p, q(< a, b, p > \in R1 \wedge < q, b, c > \in R2)\}$

Justificación: Esto debido a que en la consulta estamos pidiendo el conjunto de las tuplas *a, b, c* tal es que existen un *p* y *q* sobre las cuales *a, b, p* pertenecen a *R1* (Siendo el equivalente a proyectar los atributos *A1* y *A2* de la relación *R1*), y sobre las cuales *q, b, c* pertenecen a *R2* (Siendo el equivalente a proyectar los atributos *A2* y *A3* de la relación *R2*). Siendo que las tuplas resultantes de *R1* y *R2* sean las mismas con el operador ∧ (Siendo esta operación el equivalente del join).

Actividad No. 2

1. $\sigma_{B2 = "Andres" \ OR \ B2 = "Eduardo"}(R2)$

R//: $\{< c, d > | < c, d > \in R2 \wedge (d = "Andres" \wedge d = "Eduardo")\}$

2. $\{< a > | \exists\, b(< a,\ b > \in R1 \wedge a = 17)\}$
R//: $\pi_{A1}(\sigma_{a=17}(R1))$

3. $\{< a,\ b,\ c > | < a,\ b > \in R1 \wedge < a,\ c > \in R2\}$
R//: $\pi_{A1,A2,B2}(\sigma_{A1=B1}(R1 \bowtie R2))$

4. $R1 \bowtie_{A1<B1} R2$
R//: $\{< a,\ b,\ c,\ d > | < a, b > \in R1 \wedge < c, d > \in R2 \wedge (a < c)\}$

Actividad No. 3

1. Cree el archivo con el nombre createdb.sql con las instrucciones para crear las tablas/relaciones y tenga en cuenta

   Primero que todo, en dicho archivo createdb, se crea la base de datos con `CREATE DATABASE lab4` y sus respectivos parámetros. De allí, nos conectamos a ella con `\c lab4.`

   a. El atributo student_id inicia en 7488 con un incremento de 168 cada vez que se inserta un estudiante.

```
CREATE SEQUENCE seq_student
INCREMENT 168
START 7488;
```

   Esto se puede lograr gracias a una secuencia que empieza en 7488, e incrementa 168 por cada registro. Se usa con ayuda de un nextval, que vendría formando la clave primaria de la relación estudiante, donde por cada nuevo estudiante, toma el valor siguiente de la secuencia:

```
student_id INTEGER PRIMARY KEY DEFAULT NEXTVAL('seq_student'),
```

   b. El atributo course_id inicia en 837827 con un incremento de 23.

```
CREATE SEQUENCE seq_course
INCREMENT 23
START 837827;
```

   Tal y como en el punto anterior, el cometido se logra con ayuda de una secuencia, en este caso, empezando por 837827 con un incremento de 23. Dicha secuencia se usa en un nextval, formando la clave primaria de la relación:

```
course_id INTEGER PRIMARY KEY DEFAULT NEXTVAL('seq_student'),
```

c. El tipo de dato del atributo grade de la tabla enrolls es NUMERIC con dos decimales, superior a 1.00 y menor a 5.00

```
grade NUMERIC(3, 2) CHECK (grade BETWEEN 1 AND 5),
```

Este punto se logra en dos pasos, primero, usando `NUMERIC(3, 2),` que representa un valor numérico con 3 cifras significativas y dos cifras decimales. y a su vez, con `CHECK,` se verifica que el valor esté entre el rango 1.00 y 5.00, de lo contrario, la consulta arrojará un error por check constraint violation.

Las otras relaciones se crean de forma convencional de acuerdo al esquema expuesto.

2. Cree un archivo con el nombre insert.sql que contenga las instrucciones para insertar los datos a cada una de las tablas de la BD. Tenga en cuenta:

   a. Ejecute el script createdb.sql para llevar el esquema a PostgreSQL

   b. Cada tabla debe contener al menos 5 filas. Se recomienda el uso de generadores de datos ficticios (dummy).

   c. Se pueden considerar datos simples o inventados como "Profe 1", "Profe 2", "Curso I", "Curso II" u otros.

   Se han considerado datos simples como se indica en el inciso 3, cada relación ha quedado con 6 tuplas, a excepción de requires, que ha quedado consigo. Esto, claramente después de haber ejecutado el script createdb y estando conectado a dicha base con `\c lab4.` Las inserciones tienen la siguiente forma:

```
INSERT INTO student(name, program)
VALUES
    ('Student 1', 'Program 1'),
...
```

3. Cree un archivo con el nombre queries.sql que contenga las instrucciones para obtener registros previamente consignados en la tabla de datos.

   a. Encuentre el dept, title de los instructores registrados en la base de datos.

```
SELECT dept, title FROM instructor;
```

Con ayuda de esta sentencia, se hace el equivalente a una proyección del

álgebra relacional, obteniendo una nueva relación con todas las tuplas, pero únicamente conteniendo los atributos dept y title de Instructor.

b. Indique el nombre y programa del estudiante con student_id = 7656

```
SELECT name, program FROM student WHERE student_id = 7656;
```

Con ayuda de esta sentencia, se hace una consulta equivalente a una proyección sobre una selección del álgebra relacional, donde primero se obtienen las tuplas donde el student_id cumpla la condición planteada, y finalmente, se hace una partición de los atributos name y program, requeridos para la nueva relación.

c. Encuentre los nombres de todos los estudiantes que se han matriculado en el curso con course_id = 837873

```
SELECT name FROM student NATURAL JOIN enrolls
WHERE course_id = 837873;
```

Con ayuda de esta sentencia, se hace una consulta equivalente a una proyección sobre una selección, a su vez realizada sobre un natural join del álgebra relacional, donde primero se obtienen las tuplas de dicho join donde el course_id cumpla la condición planteada, y finalmente, se hace una partición del atributo name para la nueva relación.

d. Cree una vista llamada better_students que presente los estudiantes que obtuvieron las notas más altas por cada semestre entre los años 1900 y 2018

```
CREATE VIEW better_students AS
    SELECT * FROM student NATURAL JOIN (
        SELECT year, enrolls.semester, max, student_id
FROM enrolls INNER JOIN (
            SELECT max(grade), semester
            FROM enrolls
            WHERE year BETWEEN 1900 AND 2018
            GROUP BY semester
        ) AS max_per_semester
        ON max_per_semester.semester = enrolls.semester
AND max_per_semester.max = enrolls.grade
    ) AS best_student_per_semester;
```

Con ayuda de esta vista, quedan representados los mejores estudiantes por semestre en el conglomerado de todos los años. Es una composición de operaciones, que puede ser vista como: primero, una selección de la nota máxima (operador Gamma), y el semestre, de la relación enrolls, únicamente teniendo en cuenta las tuplas donde el año esté entre 1900 y 2018, y siendo

agrupadas por semestres. Después de eso, se hace un theta join con la misma relación enrolls, para así únicamente obtener los estudiantes con mayor nota por semestre, pues bien, en el paso anterior, todos los estudiantes aparecían en la relación resultante. De dicha relación se obtienen atributos como el año, semestre y demás, simplemente con fines informativos, que, junto con el join natural hecho entre la relación resultante y la relación student, presenta "toda" la información relevante de los estudiantes con mejor nota por semestre.

4. Cree un archivo con el nombre tandplpg.sql con los siguientes procedimientos (procedures) y disparadores (triggers)

    a. Cree uno o varios disparadores (triggers) que implemente los siguiente requerimientos para la relación enrolls

    Primero, se crea el trigger, se determina que actúe `BEFORE INSERT` en la relación enroll, para así poder comprobar los valores a ingresar antes de que se haga la operación. Se ha dispuesto de esta manera porque resulta contraproducente copiar todo el código en este apartado. Ahora, por cada fila, se va a ejecutar un procedimiento que se debe crear:

        i. Al agregar una tupla en enrolls, en caso de que la nota sea negativa, cero (0.0) o mayor de 5.00 se debe generar una excepción indicando que el valor a guardar en grade es incorrecto o invalido.

```
CREATE FUNCTION grade_excep() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.grade < 1.00 OR NEW.grade > 5.00 THEN
        RAISE EXCEPTION 'Grade must be a value between
1.00 and 5.00';
    END IF;
```

Con ayuda de este condicional, se comprueba si el atributo grade de la tupla a insertar se encuentra dentro del rango establecido [1, 5]. De lo contrario, se lanza una excepción dejando un mensaje al usuario.

        ii. Durante la actualización de un registro si el valor grade es modificado, usando RAISE NOTICE se debe presentar un mensaje indicando el cambio, si es igual al valor grade en la tabla se debe indicar que el valor no ha sido modificado. Si el grade a actualizar es negativo, cero o mayor de cinco use RAISE EXCEPTION.

```
IF NEW.grade = OLD.grade THEN
        RAISE NOTICE 'Grade has not been modified';
    ELSE
        RAISE NOTICE 'Grade has been modified TO %',
NEW.grade;
```

```
    END IF;
```

Para esto, se ha usado el mismo trigger, sólo que ya no se activa únicamente al hacer un insert a la relación enroll, sino que también lo hace al hacer un update en ella. Si la nota de la nueva tupla es igual a la nota de la tupla a reemplazar, se da el aviso al usuario de que la nota no fue modificada, de lo contrario, se avisa que si lo hizo.

b. Cree un procedimiento create_teaches que automáticamente agregue un registro a teaches. Este recibe dos argumentos un identificador de instructor instructor_id y un identificador de course_id. Se asume que ambos existen en la base de datos.

   i. Este procedimiento debe verificar que el curso exista en la oferta de cursos.

   ii. Use curse_id, sec_id, year y semester de la oferta de curso y instructor_id el para insertar en teaches.

   El procedimiento es ciertamente extenso por lo que no vale la pena colocar su código aquí, únicamente se tratará de explicar. Primero, se declaran 3 variables enteras que serán utilizadas para almacenar los resultados de las consultas del semestre, año y sección (necesarias para hacer un insert en la relación teaches) . Después de eso, se comprueba si el curso está en oferta, en caso contrario, se indica al usuario justamente eso. Si realmente lo está, se hacen las consultas anteriormente nombradas, y se hace un insert a la relación teaches, con esos valores, acompañados del id del curso y del docente recibidos como parámetro

5. Cree un archivo con el nombre drops.sql con las instrucciones necesarias para hacer el borrado de todas las tablas, secuencias, disparadores, etc, que se crearon en los anteriores puntos de este laboratorio. (No utilice la instrucción CASCADE)

   De esta sección no se hace mayor comentario, pues bien, todas las sentencias se basan en DROP. Los único que cambia como tal es el argumento, pasando de ser DROP TABLE a DROP SEQUENCE, entre otros.

   DROP TRIGGER grade_excep_trigger ON enrolls;

   DROP FUNCTION grade_excep;

   DROP VIEW better_students;

   DROP TABLE requires;

6. Comprima todos los archivos sql en un archivo LABNO1-APELLIDO1-APELLIDO2 (.zip), (.rar) o (.7z) y junto con un informe en PDF describa las soluciones realizadas en cada punto de este laboratorio. (Utilice este documento como plantilla para el informe)

Pequeña anotación: Debido a que la respectiva asignación del campus virtual solo permite archivos en formatos .pdf, nos vimos en la obligación de asignar el archivo .zip en un repositorio de Github que se encuentra en el siguiente enlace: https://github.com/Seb0927/BD_Actividad3/tree/main